SCHOOL OF INFORMATION SCIENCES AND TECHNOLOGY

THE PENNSYLVANIA STATE UNIVERSITY

dTank: A Competitive Environment for Distributed Agents

Isaac G. Councill, Geoffrey P. Morgan, and Frank E. Ritter

igc2@psu.edu gmorgan@psu.edu frank.ritter@psu.edu

Technical Report No. ACS 2004 - 1

30 March 2004

acs.ist.psu.edu

Phone +1 (814) 865-4455 Fax +1 (814) 865-6426

Applied Cognitive Science Lab, School of Information Sciences & Technology, University Park, PA 16802

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE **30 MAR 2004**	2. REPORT TYPE	3. DATES COVERED **-**
4. TITLE AND SUBTITLE **dTank: A Competitive Environment for Distributed Agents**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Office of Naval Research,800 N. Quincy Street,Arlington,VA,22217**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
The original document contains color images.

14. ABSTRACT
dTank provides an architecture and platform neutral test-bed for adversarial real-time cognitive models, and a tool for teaching cognitive modeling, and agent creation. It was inspired by Tank-Soar and ModSAF systems. This report focuses on how to use dTank with sections on controls, map-making, and its I/O, including as an example a way to provide Soar agents access.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**23**	

dTank: A Competitive Environment for Distributed Agents

Isaac G. Councill, Geoffrey P. Morgan, and Frank E. Ritter

igc2@psu.edu gmorgan@psu.edu frank.ritter@psu.edu
Applied Cognitive Science Lab
School of Information Sciences and Technology
The Pennsylvania State University
University Park, PA 16802

ACS #2004-1
30 March 2004

Abstract

dTank provides an architecture and platform neutral test-bed for adversarial real-time cognitive models, and a tool for teaching cognitive modeling, and agent creation. It was inspired by Tank-Soar and ModSAF systems. This report focuses on how to use dTank, with sections on controls, map-making, and its I/O, including as an example a way to provide Soar agents access.

Acknowledgements

This research was supported by the Office of Navy Research, contract N000140210021. Andrew Reifers and Kevin Tor have provided critical assistance in the overhaul of the dTank communication protocols. Alex Wood designed the dTank 1st person interface. Mark Cohen built the dTank/Jess API.

Table of Contents

1.0 About dTank 1
  1.1 Purpose of dTank 1
  1.2 Users of dTank 2
  1.3 dTank system requirements 2
  1.4 Similarities and Differences between dTank and related systems 2
2.0 Game Architecture 3
3.0 Interfaces 3
  3.1 Plan-view interface 4
    3.1.1 Installation, Configuration, and Startup of Plan-View Display 5
    3.1.2 Plan-View Display Controls 5
  3.2 3D Interface 6
    3.2.1 Installation, Configuration, and Startup 7
    3.2.2 Controls 7
  3.3 Logging player and agent behavior 8
  3.4 Making new maps 9
4.0 dTank API 10
  4.1 Creating a communication channel to the server 10
  4.2 Commands to the Server 11
SET TEAM 12
  4.3 Messages from the Server 13
    4.3.1 Initialization 13
    4.3.2 Server updates 13
  4.4 Agent vs. 3D API Differences 15
  4.5 Soar I/O Structure 17
5.0 dTank/Jess API 18
6.0 References 19
Appendix: Troubleshooting 1
  A.1 Common 2D Interface problems 1
    A.1.1 The application does not load. 1
    A.1.2 dTank loads, but does not run 1
  A1.2 Common 3D Interface Problems 1

1.0 About dTank

dTank was originally inspired by Tank-Soar (ftp.eecs.umich.edu/~soar/tanksoar.html), developed by Mazin As-Sanie at the University of Michigan. Tank-Soar provides an environment for Soar models to drive tanks against other models. As with Tank-Soar, the goal of dTank is to provide a test-bed for adversarial real-time cognitive models, similar in many ways to AI-based computer game opponents. dTank was developed to take advantage of the flexibility of Java graphics and networking.

Unlike Tank-Soar, dTank has both a 2D and 3D interface, with an overhead map and third-person point of view respectively. In addition, dTank provides an agent architecture-neutral interface to the game server, so that humans and artificial entities built on virtually any platform can interact within the same environment over a LAN or the internet.

1.1 Purpose of dTank

dTank is intended to serve in three distinct roles, teaching, cognitive modeling, and agent creation.

(a) as a teaching tool. dTank provides a simple, well-documented environment for experimenting with agent programming. A sample API for Soar agents is included with the dTank distribution, and instructions exist within this manual for creating interfaces for other types of agents. Our website now includes Jess agents as well.

dTank supports protocol capture from human players. Students can use dTank to study cognitive modeling techniques including model-to-data fitting, in addition to general AI applications.

(b) as a modeling tool. Due to the protocol capture facility mentioned above, as well as dTank's communication layer, dTank provides a good environment for modeling both individual and team phenomena. Any cognitive architecture that can be made to support socket communication can interact with dTank. The communication layer is general enough that virtually any theory of multi-agent communication can be tested. One could, for example, study how a user works with a social environment defined by a set of agents with known characteristics, knowledge, and behavior.

(c) as a developmental test-bed for advanced AI applications. The primary reason for the development of dTank was to create a tool to investigate the usability of distributed AI (multi-agent) systems. In particular, the ACS Lab is using dTank to inform the development of tools that help to explain the behavior (both actual and intended) of complex cognitive models to that of the modeler and human opponents of these models. This effort is aimed at improving the usability and usefulness of applied agent technologies, as well as to provide improved facilities for the validation of model behavior.

1.2 Users of dTank

Anyone is welcome to use and modify this software and its associated API. If you find dTank useful, let us know. We would enjoy hearing how dTank is being used.

1.3 dTank system requirements

dTank requires the Java Virtual Machine (JVM), version 1.4 or higher. We have tested it on Windows, Linux, and Mac OSX. The 3D client also requires Java 3D API 1.3.

dTank will load on a Mac running OSX that has the JVM, but it runs very slowly. If you understand this problem, or can help fix this, we welcome your input.

1.4 Similarities and Differences between dTank and related systems

As noted previously, dTank is inspired by Tank-Soar. Naturally, there are similarities between the two tools. Both use tanks for independent agents. Both support multiple agents in the environment simultaneously, and they both include a plan-view display. Finally, both tools can be engaging and interesting, even fun, in their own right. It is also similar to ModSAF, in that both architectures support distributed agents.

There are significant differences between dTank and these environments as well. dTank includes a 3D modified 1st-person view with slightly improved graphics and representation. dTank is not intrinsically tied to any cognitive architecture or system platform, unlike Tank-Soar. dTank supports real-time spontaneous messaging between the server and client, allowing for more realistic behaviors. Finally, dTank has time logging capabilities, which allows for data capture from human players.

2.0 Game Architecture

As shown in Figure 1, dTank's architecture has three parts. A server controls the action, the clients (Agents and Humans) connect to the server, and the API (see Section 4.0) acts as an interpreter between the server and clients. The server is designed to support a number of users, both humans and agents. We have currently tested only two of each.

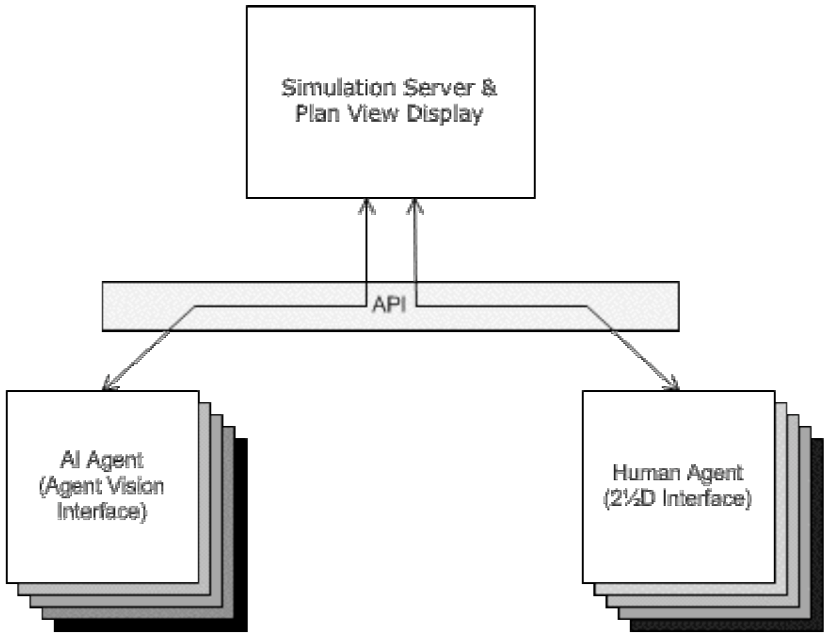


Figure 1. Architecture of the dTank simulation.

This architecture allows players to connect through sockets, which allows connections over networks, with loop-back interfaces, or on the same machine.

3.0 Interfaces

The plan view interface is intended to be used as a debugging environment; it also provides a plan view of the game-play.

Both agents and the server interpret the game as a 2D world. The 3D interface used by the human player is an interpretation of the 2D world. Although agents live in a 2D world, they have a vision component and are limited in seeing only what a human could see with the 3D interface (a small step towards what has been called fog of war). Therefore, although the 2D plan view display is available to the user; it use should be limited to demonstration work with agents, and debugging; it should not (and has not been enabled to) be used for modeling human behavior. This 2D interface eases debugging by allowing the modeler to see tank actions and determine if those actions make sense or if an agent needs revision.

3.1 Plan-view interface

The plan-view interface is composed of two large panels, the Game panel and the menu panel. The game panel displays the map, the tank's bearing, the tank turret's bearing, and the current position of the tank on the map.

The menu panel contains several sub-panels. Figure 2 provides a sample screenshot of the 2D interface with comments. The sub-panels include an [Enter] and [Exit] button. These are in-application commands. [Enter] implies that you wish your tank to enter the game environment; [Exit] implies you wish your tank to leave the environment. The [Documentation] button pulls up this manual. The [Client] info button displays network connection information about you, the client. The [Quit] dTank button is used to leave the dTank application.

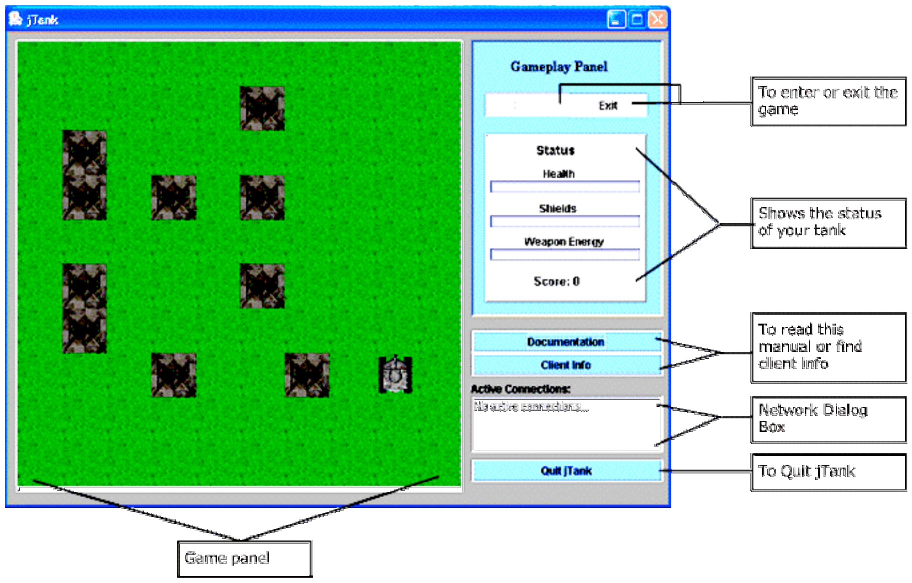


Figure 2. Interface guide for the plan-view display.

The game panel displays the environment of dTank. The terrain of dTank is of two types, grassy plain and impassable rock. A tank cannot overcome a rock barrier, but has to maneuver around it. Rocks are used as barriers by the strategic player, as fire cannot pass through a rock barrier. In Figure 3, possible paths are shown for a subset of the terrain shown in Figure 2.

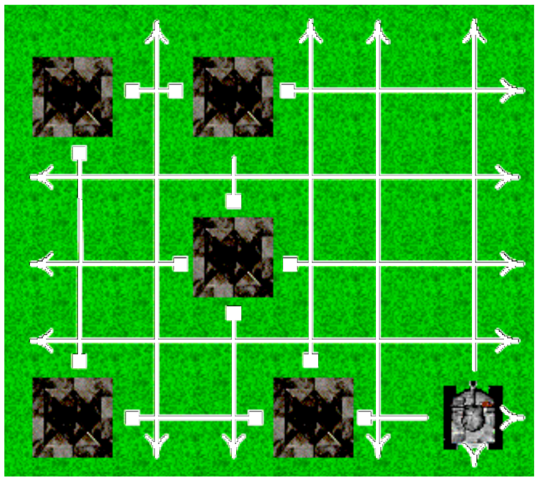


Figure 3. Tank Pathfinding

3.1.1 Installation, Configuration, and Startup of Plan-View Display

To run the Plan-View Display, you will need to install the JVM 1.4. Download the dTank Game-Server (the Plan-View is incorporated) code and resources and unpack all files to a single directory. If you are on Windows, you can start the Game-Server by double-clicking on the dTank-2.0.7.jar file; otherwise, enter a command shell and type 'java dTank-2.0.7' from the Game-Server directory. You can modify the Config.txt file to specify server initial port, base agent ports, and the name of the map file.

3.1.2 Plan-View Display Controls

The controls of dTank are not complex. However, it is important to note that all tank movement commands, as well as firing, are mutually exclusive. In other words, you cannot move in two directions at once, or fire while moving in any direction. Also, while shields can be up while you move, you cannot fire with shields on. However, aiming the tank's turret barrel can be performed at any point, during the execution of a keyboard command or not.

Controls are not operational for a tank until it completely enters the game (becomes opaque). Figure 4 lists the controls and their in-game effects.

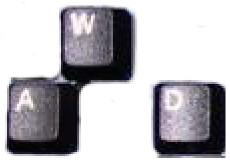
Control(s)	In-game Effect(s)
	[A] turns counter-clockwise; [D] turns clockwise; [W] is move forward. The tank cannot go in reverse.
	[X] activates the shields, for a short period of time. You do not need to hold down this button. Shields cost energy.
	[Spacebar] fires a shot from the tank. Holding down the spacebar allows multi-firing. If your target is moving, spread-fire by re-aiming the barrel while multi-firing.
	Left-clicking the mouse turns the turret towards the direction clicked.

Figure 4. Controls for the Plan-View Display

3.2 3D Interface

There is a 3D interface (2.5D to be precise). The view is a 3D rendering of the two-dimensional dTank game world. There are two reasons for its creation:

(a) To align the human dTank experience as closely as possible with agents' experience. A dTank agent can only see objects within about a 100 degree angle centered on its turret. If agents are to be modeled after human users (one of the reasons for creating dTank), it is useful to collect data from humans who operate in similar conditions to the agents (for more information on 3D and Plan-View API differences, see Section 4.4).

(b) For fun. Playing the 3D interface is meant to provide more of a challenge to human players. It is no fun playing against agents with limited sensing abilities compared to a complete view through the plan-view display. The 3D client also provides a simple means to access the dTank server via the server's network interface.

3.2.1 Installation, Configuration, and Startup

To run the 3D client, you will need to install the Java 3D API 1.3 or higher. Download the dTank 3D client code and resources and unpack all files to a single directory. If you are on Windows, you can start the 3D client by double-clicking on the dTank3D.bat file; otherwise, enter a command shell and type 'java Tank3D' from the 3D client directory. This will bring up the interface and connect to a dTank server if one is found. You can modify the Config.txt file to specify server location, port, and the agent's name.

3.2.2 Controls

Controls for the 3D interface are very similar to those of the Plan-View Display. However, the mouse is not enabled for use in the 3D interface.

Table 1. 3D interface controls.

Movement	**Turret Actions**
W – move forward	**Z** – Rotate turret counter-clockwise
A – turn counter-clockwise	**X** – Rotate turret clockwise
D – turn clockwise	**Space** – Fire
S – To turn on Shields	

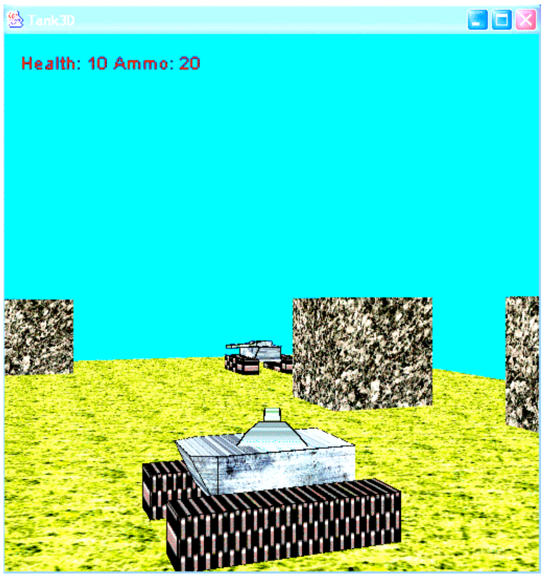


Figure 5. A screenshot of the dTank 3D interface.

3.3 Logging player and agent behavior

To create a detailed log file of your dTank sessions (human or Soar interface 2.0.7), you will simply need to add or modify a line in your clients' Config.txt file. Make sure your Config.txt contains the following:

```
LOGGING 1
```

This will cause the server to log all communications between the client and server to a file named after the agent id. The file will be located in the directory in which dTank was started on the server machine (see Table 2 for example log format).

Of course, you may not be using pre-packaged APIs. If you are using your own API to the dTank server you will need to send commands to the server that will cause the server to start and stop logging. These commands are:

```
startLogging|
```

```
stopLogging|
```

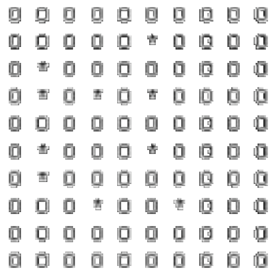
The logging commands can be sent at any point during game play. To log entire sessions simply send the `startLogging` command immediately after agent connection.

Table 2. Portion of an example log file, including time (in ms), sender, and message.

```
1083247478749 SERVER ACK:nok:I245:|
1083247478968 CLIENT rotate|1|I248
1083247478968 SERVER ACK:ok:I248:|
1083247478968 SERVER INPUT:blocked:|bye
1083247479015 CLIENT rotate|1|I250
1083247479015 SERVER ACK:nok:I250:|
1083247479109 SERVER ACTION:2Dhuman:moveForward:|bye
1083247479171 SERVER VISUAL:Stone:X|6:Y|0:Stone:X|1:
Y|2:Stone:X|2:Y|4:Stone:X|5:Y|8:Tank:ID|2Dhuman:Color
|grey:X|8:Y|6:Orient|3:TurretRot|5.5566920060369425:
ShieldStat|no:moving|true:rotating|false:|bye
1083247479437 CLIENT rotateTurret|5.9614307724
1083247479765 SERVER INPUT:clear:|bye
1083247479796 SERVER
EVENT:turretRot:5.910121179565796:|bye
1083247480015 CLIENT rotateTurret|5.9614307724
```

3.4 Making new maps

You can create new maps for dTank. The environment map for dTank is in the same directory as the dTank JAR file and is called “world.map”. Figure 6 demonstrates what “world.map” looks like for the examples provided in this manual.



```
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 * 0 0 0 0
0 * 0 0 0 0 0 0 0 0
0 * 0 * 0 * 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 * 0 0 0 * 0 0 0 0
0 * 0 0 0 0 0 0 0 0
0 0 0 * 0 0 * 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

Figure 6. Map of the example world shown in Figure 2.

The map is a 10 by 10 matrix of cells. Each cell can have one of only two possible values: ‘0’ or ‘*’. A ‘0’ represents a grassy plain; a ‘*’ an impassable rock. Figure 6 superimposes the world.map matrix onto the plan view interface.

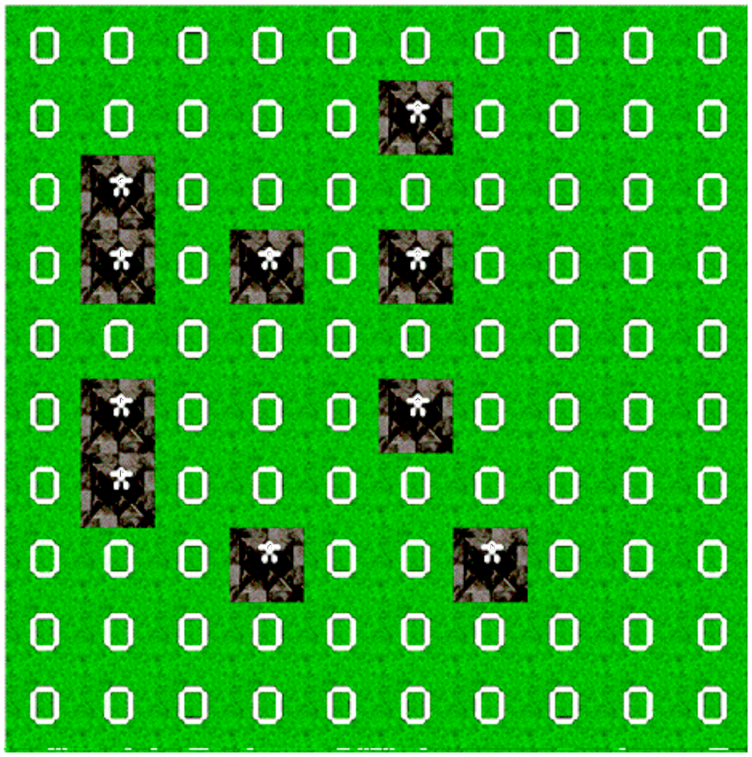


Figure 7. Map file from Figure 6 superimposed on the game panel from Figure 2.

To create new maps is not difficult. Copy “world.map”; edit the map with Emacs by replacing ‘*’ with’0’ and vice-versa as appropriate. Save the new map as world.map, and restart dTank.

4.0 dTank API

As mentioned in Section 2.0, dTank’s API acts as an interpreter between the server and its clients. We explain the API and how to use this API in this section.

4.1 Creating a communication channel to the server

For an agent to join the dTank server, it needs to do the following steps.

(a) Connect to the dTank server socket and send the following: "agent join"

(b) Server responds with the following: "join|<<int>>", where <<int>> is a port number on the server host

(c) Close the connection to the server socket and open a new connection to the port number specified in the join message. Before connecting, pause for half a second or so to allow time for the server to create a new listener-socket.

Your agent is now connected and you will receive an initialization string (see Section 4.3.1) over the socket.

4.2 Commands to the Server

Commands are sent by the client to the server, so it can update the state of the world. Most commands to the server should incorporate unique command IDs so that acknowledgement (ACK) functionality works properly (see Section 4.3.2). Where necessary, the command ID will be noted by "<id>" in the following syntax descriptions.

MOVE

The tank moves forward one cell when this command is acknowledged by the server

Format: moveForward||<id>

TURN

Turns the tank 90 degrees clockwise or counter-clockwise.

Format: rotate|<<int>>|<id>,

For this command, <<int>> may be 1 for clockwise or 0 for counter-clockwise.

ATTACK

The tank fires in the direction faced by the turret upon successful acknowledgement of the command by the server.

Format: attack||<id>

ROTATE TURRET

Rotates the tank's turret to an angle specified in radians, where 0 is north. See Figure 8 as well.

Format: rotateTurret|<<double>>|

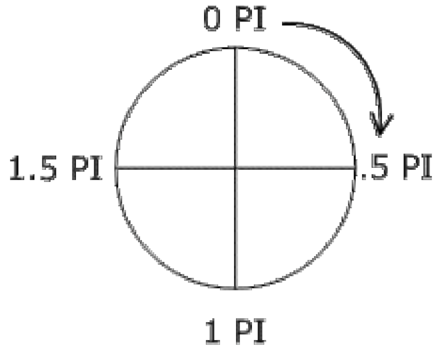


Figure 8. Radian measures.

RAISE SHIELDS

The tank raises its shields when this command is acknowledge by the server.

Format: raiseShields||<id>

SCAN

The agent requests a scan for detailed stats on a specified tank within its visual field.

Format: scan|<<string ?tank-id>>|<id>

RADIO

The agent communicates to other tanks in the dTank world. There are three message types: (a) broadcast messages can be used to post the message to all other tanks in the dTank world, (b) unicast messages are agent-to-agent communications where there is only one-recipient, and (c) team messages are sent to all agents on the same team as the sender.

Format: RADIO|<<string broadcast|team>>|<<string>>,

Format: RADIO|unicast|<recipient-id>|<<string>>

SET TEAM

The agent announces its team allegiance. This command can be sent to the server at any point in the game, but is generally most useful at startup.

Format: setTeam|<<string>>

SET NAME

The agent announces its name. This is an optional field used to identify different tanks, rather than by ID number. This command can be sent to the server at any point in the game, but is generally most useful at startup.

Format: setName|<<string>>

START LOGGING

This tells the server to start a log for this client. This can be used to create a log for humans or agents. For more information, see Section 3.3.

Format: startLogging|

STOP LOGGING

This tells the server to stop logging this client's actions. For more information, see Section 3.3.

Format: stopLogging|

4.3 Messages from the Server

During game-play, agents will receive un-requested messages from dTank about the actions of visible players and game status.

4.3.1 Initialization

You will get an initialization string when your agent connects to the dTank server for the first time.

Format: InitialSettings:Name|<<string>>:Color|<<string>>:Health|<<int>>:Weapon|<<int>>: Shields|<<int>>:X|<<int>>:Y|<<int>>:Orient|<<int>>:TurretRot|<<double>>

4.3.2 Server updates

During game-play, your agent will receive various message types from the dTank server.

STATUS

Periodically (about every 2 seconds) each agent will receive a status string that informs it of its current attributes

Format: STATUS:Health|<<int>>:Weapon|<<int>>:Shields|<<int>>:X|<<int>>:Y|<<int>>:
Orient|<<int>>:TurretRot|<<double>>:ShieldsStat|<<string up|down>>

VISUAL

About every 2 seconds each agent is sent a string containing information for all objects that your agent can currently see. This will include all objects in a 100 degree angle centered on the agent's turret orientation. Agents cannot see objects that are obscured by other tanks or stones.

Format: VISUAL:{Tank:ID|<<string>>:X|<<int>>:Y|<<int>>:Color|<<string>>:
ShieldsStat|<<string>>:Orient|<<int>>:TurretRot|<<double>>:}*{Stone:X|<<int>>:Y|<<int>>:}*

Please note that the '*' indicates there can be more than one of them in the string sent. Two stones, for example, can be recorded in one message.

Example: Two stones in view.

```
VISUAL:Stone:X|5:Y|5:Stone:X|2:Y|2:
```

Example 2: 1 tank in vew

```
VISUAL:TankID|thisTank:X|3:Y|4:Color|Cyan:ShieldStat|Up:Orient|3:TurretRot|3.14:
```

SCAN

If an agent successfully request a scan of another tank in the game, the server will send a scan message. This gives important and more detailed information about rival tanks that cannot be discovered with the visual message.

Format: SCAN:<<string ?tank-id>>:Health|<<int>>:Weapon|<<int>>:Shields|<<int>>

RADIO

If another agent sends a message to your agent, you will receive a radio string.

Format: RADIO:<<string ?tank-id>>:<<string ?message>>

ACK

The server will acknowledge all commands sent to dTank as either acceptable (ok) or not acceptable (nok). To use this functionality, your agent interface must send command ids with every command string as described in the output section.

Format: ACK:<<string ?id>>:<<string ok|nok>>

INPUT

When your agent's environmental status changes (possible to move, not possible to move, reloading cannon) your agent will be notified. Agents may not move forward or turn while moves are not possible. Likewise, agents may not fire their cannon while reloading.

Format: INPUT:{<<string blocked|clear>>}|{reloading|<<string yes|no>>}

EVENT

When your agent is hit, killed, or moves its turret to a new orientation, an event message will be sent.

Format: EVENT:<<string hit|died>>

Format: EVENT:turretRot:<<double ?orientation>>

ACTION

When tanks that your agent can see engage in some action, your agent will be notified with an action message.

Format: ACTION:<<string ?tank-id>>:{moveForward|}|{raised-shields}|{lowered-shields}|{rotate|<<string clock|c-clock>>}|{rotateTurret|<<double>>}|{fired| <<double>>}

4.4 Agent vs. 3D API Differences

One of the goals of dTank is to provide a testing environment in which agents and humans can interact on equal footing. One of the requirements imposed by this goal is that both synthetic and human players should have as close as possible to the same sensory input from the game environment. For instance, if humans are able to view the entire game board along with the positions of all tanks while agents can only see other tanks within their visual field, human players would have an unfair advantage over their synthetic opponents. Upon first glance over the API specifications it appears that such iniquity does in fact occur. The 3D (human) API keeps clients up to date with the complete game state, including the positions and orientations of all game objects. In contrast, the agent API only transmits information on a can-know basis; that is, agents are only told about those things that they can immediately see or hear.

The reason for this disparity is that sensory filtering takes place at different stages of each interface (see Figure 9). For agents, game events are explicitly filtered at the server based on the state of the perceiving agent. For humans using the 3D interface, all game object information is sent to the client, but the client presents only the human-controlled tank's visual field of the game world. In this way, an efficient 3D rendering environment can be maintained while controlling sensory filtering implicitly as the player navigates the 3D world.

However, as at the time of this writing the 3D interface is a work in progress. It is not complete in that its view is not fully aligned with the agent view. For example, agents can get other tank's statistics (such as health, shields, etc.) and this is not yet displayed in the 3D interface, although possible to acquire using a command line (though this is not recommended).

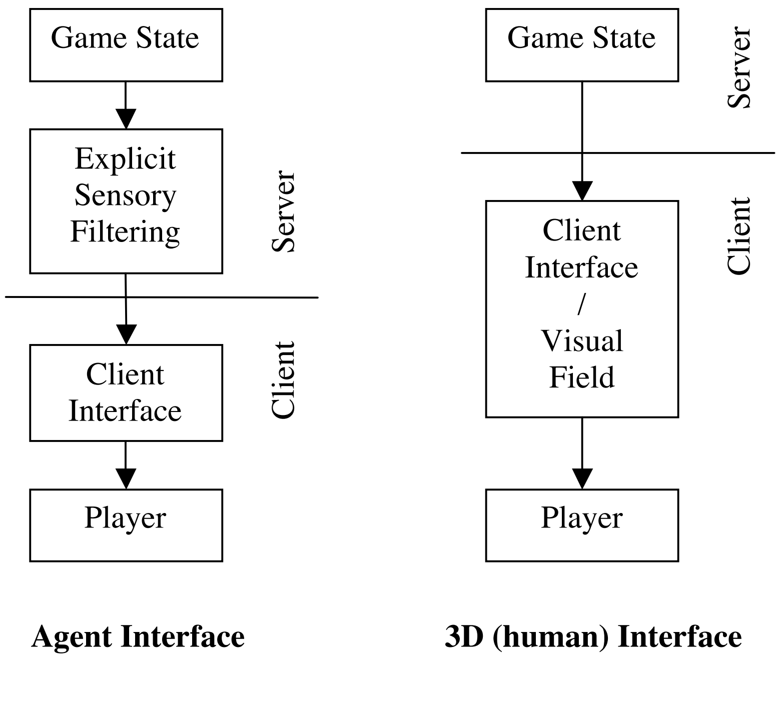


Figure 9. Sensory filtering takes place at different stages of the agent and human interfaces.

4.5 Soar I/O Structure

It will be helpful to keep in mind a few things when programming soar agents using the provided soar interface in Table 3.

The I/O link is very dynamic! An agent should not depend on the input-link to be a stable source of current information. The information on the input-link (status, visuals, events, actions) only stay on the input link for one cycle after the information is received from the server. That is, an agent will have substructure on the ^io.input-link.status node for only one cycle every 2 seconds or so. Also, it is possible to have conflicting information on the I/O link if, for instance, a visual message and real-time action are received on the same cycle.

Of course, agents will need to keep a stable memory structure containing a memory of status, location, etc. as well as what it knows (or thinks it knows) about the world and other tanks. For that reason, it will be best to monitor the input link for new information and update stable memory structures based on new input. The agent should then test the stable memory structure for status and enemy information and the input link for the move-possible and pending-actions attributes before acting. See the code examples provided with the dTank distribution for more details.

When starting to program dTank agents, do not worry about the real-time actions at first. Agents can get by without them, but they may be acting on information outdated by over 2 seconds. Once you are comfortable with the interface, you can start programming your agents to pay attention to specific real-time events depending on your situation. For example, if an agent is attacking two enemy tanks, the agent may want to pay particular attention to the tanks' movements and turret rotations. If both tanks decide to target the agent, it may want to react immediately! The downside is that the more an agent pays attention to, the more of its time that is spent updating information rather than reasoning about activities or taking other actions. It is up to you to decide the appropriate balance of activity for your agent.

Table 3. Soar I/O Structure.

```
^io
 ^input-link
  ^dtank
   ^id <<string>>
   ^tank-color <<string>>
   ^hit yes
   ^died yes
   ^status
    ^health <<int 1-10>>
    ^weapon-energy <<int 1-20>>
    ^shields-energy <<int 1-10>>
    ^x <<int 0-9>>     # ranges may change
    ^y <<int 0-9>>
    ^orientation <<int 1-4>>
    ^turret-rotation <<double>>
    ^shields-status yes | no
    ^move-possible yes | no
    ^cannon-ready yes | no
      ^visual-field
    ^new-data yes | no
    ^tank
     ^id <<string>>
     ^color <<string>>
     ^shield-status up | down
     ^x <<int 0-9>>
     ^y <<int 0-9>>
     ^orientation <<int 1-4>>
     ^turret-rotation <<double>>
    ^stone
     ^x <<int 0-9>>
     ^y <<int 0-9>>
   ^scan-results
    ^tank-id <<string>>
    ^health <<int 1-10>>
    ^weapon-energy <<int 1-20>>
    ^shields-energy <<int 1-10>>

  ^realtime-actions
   ^move
    ^tank-id <<string>>
   ^turn
    ^tank-id <<string>>
    ^direction clockwise | counter-clockwise
   ^rotate-turret
    ^tank-id <<string>>
    ^angle <<double>>
   ^fire
    ^tank-id <<string>>
    ^direction <<double>>
   ^raise-shields
    ^tank-id <<string>>
   ^lower-shields
    ^tank-id <<string>>

  ^pending-actions
   ^move
    ^status pending | ok | nok
   ^turn
    ^status pending | ok | nok
   ^fire
    ^status pending | ok | nok
   ^raise-shields
    ^status pending | ok | nok
   ^scan
    ^status pending | ok | nok
   ^send-message
    ^status pending | ok | nok

  ^radio
   ^message
    ^timetag <<int>>
    ^sender-id <<string>>
    ^body <<string>>

^output-link
  ^dtank
   ^move
    ^status complete
   ^turn
    ^direction clockwise | counter-clockwise
    ^status complete
   ^fire
    ^status complete
   ^rotate-turret
    ^angle <<double>>
    ^status complete
   ^raise-shields
    ^status complete
   ^scan
    ^id <<string>>
    ^status complete
   ^message
    ^recipient <<string>>
    ^type broadcast | unicast | team
    ^body <wme-id>
    ^status complete
```

5.0 dTank/Jess API

The dTank/Jess API is a Java Library that makes it possible to write intelligent tanks for dTank using Jess, a tool for building expert systems. Mark Cohen, at Lock Haven University, wrote the dTank/Jess API. Ernest Friedman-Hill of Sandia National Laboratories wrote Jess, and it is available for academic use at no cost. To obtain the special academic license, contact Sandia's Technology Transfer Office. To learn more about Jess and how to obtain an academic license, see the Jess website (herzberg.ca.sandia.gov/jess/).

You can download the dTank/Jess API from the ACS website (acs.ist.psu.edu/dTank). After installing Jess and downloading the dTank/Jess API, create a directory named `dTankJess` and extract the dTank/Jess zip file into this directory.

The dTank/Jess API contains examples that illustrate how to create an intelligent tank in dTank. To run these examples, or your own tanks, first execute Jess. Make sure that the dTankJess jarfile is in Jess's classpath. To add the jarfile to Jess's classpath, assuming that the Jess jarfile is installed in `\jess` and the current directory is the dTankJess installation folder - use this command:

```
java -classpath \jess\jess.jar;dist\dTankJess.jar jess.Main
```

After Jess is running, start up the dTank server and you will be ready to try out some of the example Jess tanks. To load the simple tank example, type the following from the Jess command line:

```
(batch examples\SimpleTank.clp)
```

For more information about the details of the Jess/dTank API please visit www.lhup.edu/mcohen/dTank/dTankJess.htm.

6.0 References

As-Sanie, M. (2003). *TankSoar,* from http://www.eecs.umich.edu/~soar/projects.html

Calder, R.B., Smith, J.E., Courtemanche, A. J., Mar, J. M. F., & Ceranowicz, A. Z. (1993). *ModSAF behavior simulation and control*. Orlando, Florida.

Appendix: Troubleshooting

This is an evolving document. Please inform Geoff Morgan (gmorgan@psu.edu) or Frank Ritter (Frank-Ritter@psu.edu) of issues we have not yet documented.

A.1 Common 2D Interface problems

A.1.1 The application does not load.

Although the .jar file is on my system, the program half-starts and then stops loading.

Using the command prompt, attempt to load the application (java –jar dTank-*version*.jar). Note the error messages that arise. If they seem to relate to Sun's image classes, you do not have all the images you need.

A.1.2 dTank loads, but does not run.

The 2D interface comes up, but the tank does not seem to load or loads slowly.

Are you using Mac OSX? If you are, dTank does not currently support the Mac OSX edition of the JVM (If you can help fix this, great!). This problem should disappear if you use a PC or Linux machine.

A1.2 Common 3D Interface Problems